REMARKS

In the Office Action, the Examiner rejected claims 1-7 and 10-41 under 35 USC §103. The claims have been amended to further clarify the subject matter regarded as the invention. These objections and rejections are fully traversed below. Claim 2 has been cancelled. Claims 1, 3-7 and 10-41 remain pending.

Reconsideration of the application is respectfully requested based on the following remarks.

REJECTION OF CLAIMS UNDER 35 USC §103

In the Office Action, the Examiner objected to claims 10-17, 29-33, and 36-41 as being dependent upon a rejected base claim, but would be allowable if rewritten in independent form including all of the limitations of the base claim and any intervening claims. Applicant will amend the allowable claims accordingly. In addition, the Examiner rejected claims 1, 3-7, 18-28, 34 and 35 under 35 USC §103 as being unpatentable over Anderson et al, U.S. Patent No. 5,448,735, ('Anderson' hereinafter) in view of Munro (Writing DLLs for Windows using Visual Basic, part 1). This rejection is fully traversed below.

Claim 1 relates to DLLs. Specifically, claim 1, as previously amended, recites, in relevant part, "wherein each of the code modules responsible for calling a next one of the code modules in the chain includes a reference to the next one of the code modules in the chain, wherein an address in memory at which the next one of the code modules in the chain is loaded is associated with the reference to the next one of the code modules in the chain, wherein the chain does not include a main program and wherein building a chain enables the one or more code modules to execute without requiring a main program responsible for calling the one or more code modules." Thus, the claimed invention enables DLLs to be called without being called by a main program. As a result, the time that is typically required to call each code module and return to the main program is eliminated.

Anderson neither discloses nor suggests the invention of claim 1. Specifically, Anderson neither discloses nor suggests "wherein each of the code modules responsible for calling a next one of the code modules in the chain includes a reference to the next one of the code modules in the chain, wherein an address in memory at which the next one of the code modules in the chain is loaded is associated with the reference to the next one of the code modules in the chain." (Emphasis added). Anderson merely discloses the use of pointers to reference code modules. Anderson neither discloses nor suggests associating an address in memory at which the next one of the code modules in the chain is loaded with the reference to the next one of the code modules in the chain. Moreover, even if a pointer is interpreted to be a variable that contains an address of a location in memory, it is important to note that Anderson's system merely links code modules, not DLLs.

The Examiner admits that Anderson does not teach that the one or more code modules are one or more DLLs. The Examiner seeks to cure the deficiencies of Anderson with Munro.

Munro is a tutorial for writing DLLs for Windows using Visual Basic. See Title. In this tutorial, Munro discloses that "DLLs are special libraries that load into memory only once, at run-time, and can thereafter be used by many programs." See page 1, lines 31-33. The tutorial indicates that a main module is implemented. See page 3, lines 28-42. The tutorial further indicates that, "[w]hile it has its own data segment, a DLL does not have a stack. Unlike DOS applications, in which the stack segment and the data segment are the same, a DLL shares the stack segment with the calling program." See page 4, lines 25-28. A DLL "has its own data segment, which is allocated by Windows when the DLL is loaded..." See page 6, lines 28-29. A DLL procedure is responsible for preserving the old DS and moving AX into DS for use by the program. See page 6, lines 30-37. "A DLL is similar in structure to an executable program under Windows. It is created by using the linker to pull together all the various modules. During the process of linking, a definition file is used to specify which files are exports..." See page 7, lines 27-30. The DLL's properties include a STUB. See page 7, lines 41-43.

Munro describes a conventional DLL system as set forth in the background section of Applicant's specification. Specifically, Munro indicates that a main module is implemented, and the use of a linker is implemented to pull together the various modules. Moreover, the DLLs are described as including a STUB. When a DLL is called, the DLL preserves the old

DS, as is typically performed via a subroutine called by a main program. As such, Munro teaches away from creating a "chain" of code modules (DLLs) for execution, as recited in the pending claims.

As set forth on page 2, lines 3-15 of the background section of Applicant's specification, "A DLL is an executable file that traditionally cannot be run independently. In other words, a DLL can only be run by being called from another executable file. This is typically accomplished through subroutine calls. For instance, the executable for the main program may be compiled with a library of "stubs" which allow link errors to be detected at compile-time." The disadvantages of performing subroutine calls are described on page 2, lines 9-15, which include the amount of overhead involved in calling each subroutine and returning to the main program. Thus, this prior art manner of calling DLLs is extremely inefficient.

Neither of the references, separately or in combination, addresses the problems associated with calling DLLs or a motivation to reduce the required execution time of DLLs. More particularly, it is important to note that neither of the cited references discloses or suggests the disadvantages of calling DLLs from a main program. Similarly, neither of the cited references discloses or suggests a solution such as that claimed. As such, there fails to be a motivation to combine the cited references. Specifically, Anderson discloses modules that are linked to one another, but does not disclose a main program. In contrast, Munro discloses the use of a main program to call DLLs, in accordance with standard processes. These two different ways of executing code are generally mutually exclusive. Therefore, it would be unnecessary and therefore non-obvious to combine these two different references, particularly since neither of the references suggests a reason to combine them (e.g., a problem to be solved). As such, Applicant respectfully submits that the claimed invention would be non-obvious and therefore patentable over the cited references.

The above arguments are equally applicable to independent claims 19, 22 and 25. Dependent claims 2-7, 18, 20, and 23-24 depend from one of the independent claims and are therefore patentable for at least the same reasons. However, the dependent claims recite additional limitations that further distinguish them from the cited references. Hence, it is submitted that the dependent claims are also patentable over the cited references.

For instance, with respect to claims 6, 7, and 21, the Examiner admits that Anderson does not explicitly teach determining one or more code modules to be executed to complete

configuration of a hardware interface of a router. Anderson fails to disclose configuring a hardware interface of a router. In fact, while Anderson discloses tasks that are to be executed by a processor, Anderson neither discloses nor suggests a method for configuring a device such as a router in the manner claimed. Applicant respectfully submits that it would not have been obvious to apply the task organization features of Anderson to configure a hardware interface of a router in the manner claimed. Accordingly, Applicant respectfully submits that claims 6, 7 and 21 are allowable over the cited art.

Similarly, with respect to claims 18 and 20, the Examiner admits that Anderson does not explicitly teach associating one of the one or more code modules with a hardware interface to identify a starting point for execution upon occurrence of an interrupt. Col. 9, lines 52-68 discloses DSP devices capable of being used for certain tasks. However, Anderson fails to disclose or suggest using such tasks to configure an interface of one of these devices or to execute the modules upon occurrence of an interrupt in the manner claimed. Accordingly, Applicant respectfully submits that claims 18 and 20 are allowable over the cited art.

In the Office Action, the Examiner rejected claim 5 under 35 USC §103 as being unpatentable over Anderson in view of Munro and further in view of Crick et al, U.S. Patent No. 5,781,797 ('Crick' hereinafter). This rejection is full traversed. Crick fails to cure the deficiencies of the primary references. Accordingly, Applicant respectfully submits that claim 5 is patentable over the cited references.

In the Office Action, the Examiner rejected claims 27-28 and 34-35 under 35 USC §103 as being unpatentable over Anderson in view of Munro and further in view of Mattson Jr., U.S. Patent No. 6,317,870, ('Mattson' hereinafter). This rejection is fully traversed below.

Mattson fails to cure the deficiencies of the primary references. In fact, Mattson relates to "optimization of inter-module procedure calls." See title. Thus, one procedure calls another procedure, and presumably must return to the procedure that called it upon completion of its execution. In other words, Mattson neither discloses nor suggests a chain as claimed (or disclosed in Applicant's specification and illustrated in FIG. 1 of Applicant's specification). Specifically, col. 5, lines 43-65 of Mattson disclose an intermodule call that calls an import stub. The call instruction is modified to include a call with an offset of the program counter plus the displacement to the import stub, as well as the displacement from

the import stub to the function being called. In other words, an address at which the function has been loaded is not directly identified. By definition, a table may include one or more entry. However, the call of Mattson is a single call instruction rather than a "branch table" (which is capable of including more than one branch or jump instruction). Moreover, Mattson merely discloses an optimized manner of implementing a procedure call. As set forth in Applicant's background section, a DLL is typically called via subroutine calls through the use of "stubs." The disadvantages of such an implementation (e.g., overhead in calling a procedure) are set forth on p. 2, lines 3-15 of Applicant's specification. Since an inter-module procedure call would require returning to the calling procedure, similar to a main program subroutine call as described in the Background section of Applicant's specification, the combination of the cited references would fail to achieve the desired result. Moreover, Mattson disclose patching call sites during execution rather than at load time. See col. 2, lines 48-60. As a result, call sites which are never executed will not be patched. Accordingly, Applicant respectfully submits that the claims are patentable over the cited art. The dependent claims recite additional limitations that further distinguish them from the cited references.

Based on the foregoing, it is submitted that the claims are patentable over the cited references. The additional limitations recited in the independent claims or the dependent claims are not further discussed as the above discussed limitations are clearly sufficient to distinguish the claimed invention from the cited references. Thus, it is respectfully requested that the Examiner withdraw the rejection of claims under 35 USC §103.

SUMMARY

If there are any issues remaining which the Examiner believes could be resolved through either a Supplemental Response or an Examiner's Amendment, the Examiner is respectfully requested to contact the undersigned attorney at the telephone number listed below.

Applicants hereby petition for an extension of time which may be required to maintain the pendency of this case, and any required fee for such extension or any further fee required in connection with the filing of this Amendment is to be charged to Deposit Account No. 50-0388 (Order No. CISCP125).

Respectfully submitted,
BEYER, WEAVER & THOMAS, LLP

Elise R. Heilbrunn
Reg. No. 42,649

BEYER, WEAVER & THOMAS, LLP
P.O. 70250
Oakland, CA 94612-0250
(510) 663-1100